Upgrade Your SSH Key to Ed25519

Risan Bagja Follow

Jan 9, 2018 · 4 min read

Upgrade your SSH key! Photo by Matt Artz on Unsplash

If you're a DevOps engineer or a web developer, there's a good chance that you're already familiar and using the SSH key authentication on a daily basis. Whether it's for logging into the remote server or when pushing your commit to the remote repository. It provides us with better security than the traditional password-based authentication.

But, when is the last time you created or upgraded your SSH key? And did you use the latest recommended public-key algorithm? If it was more than five years ago and you generated your SSH key with the default options, you probably ended up using RSA algorithm with key-size less than 2048 bits long.

Check Available SSH Keys on Your Computer

To check all available SSH keys on your computer, run the following command on your terminal:

```
for key in ~/.ssh/id_*; do ssh-keygen -l -f
"${key}"; done | uniq
```

Your SSH keys might use one of the following algorithms:

- **DSA**: It's unsafe and even no longer supported since OpenSSH version 7, you need to upgrade it!
- ⚠ **RSA**: It depends on key size. If it has 3072 or 4096-bit length, then you're good. Less than that, you probably want

to upgrade it. The 1024-bit length is even considered unsafe.

- **ECDSA**: It depends on how well your machine can generate a random number that will be used to create a signature. There's also a trustworthiness concern on the NIST curves that being used by ECDSA.

- ✅ **Ed25519**: It's the most recommended public-key algorithm available today!

Some Ed25519 Benefits

The Ed25519 was introduced on OpenSSH version 6.5. It's the EdDSA implementation using the Twisted Edwards curve. It's

using elliptic curve cryptography that offers a better security with faster performance compared to DSA or ECDSA.

Today, the RSA is the most widely used public-key algorithm for SSH key. But compared to Ed25519, it's slower and even considered not safe if it's generated with the key smaller than 2048-bit length.

The Ed25519 public-key is compact. It only contains 68 characters, compared to RSA 3072 that has 544 characters. Generating the key is also almost as fast as the signing process. It's also fast to perform batch signature verification with Ed25519. It's built to be collision resilence. Hash-function

collision won't break the system.

Are you ready to switch to Ed25519?

Generating Ed25519 Key

You can have multiple SSH keys on your machine. So you can keep your old SSH keys and generate a new one that uses Ed25519. This way you can still log in to any of your remote servers. Then slowly replace the authorized key on your remote servers one by one with the newly generated Ed25519 public-key.

Open up your terminal and type the following command to generate a new SSH key that uses Ed25519 algorithm:

Generate SSH key with Ed25519 key type

You'll be asked to enter a passphrase for this key, use the strong one. You can also use the same passphrase like any of your old SSH keys.

- `-o` : Save the private-key using the new OpenSSH format rather than the PEM format. Actually, this option is implied when you specify the key type as `ed25519`.

- `-a` : It's the numbers of KDF (Key Derivation Function) rounds. Higher numbers result in slower passphrase verification, increasing the resistance to brute-force password cracking should the private-key be stolen.

- `-t` : Specifies the type of key to create, in our case the Ed25519.

- `-f` : Specify the filename of the generated key file. If you want it to be discovered automatically by the SSH agent, it must be stored in the default `.ssh` directory within your home directory.

- `-C` : An option to specify a comment. It's purely

informational and can be anything. But it's usually filled with `<login>@<hostname>` who generated the key.

Adding Your Key to SSH Agent

You can find your newly generated private key at `~/.ssh/id_ed25519` and your public key at `~/.ssh/id_ed25519.pub`. Always remember that your public key is the one that you copy to the target host for authentication.

Before adding your new private key to the SSH agent, make sure that the SSH agent is running by executing the following command:

```
eval "$(ssh-agent -s)"
```

Then run the following command to add your newly generated Ed25519 key to SSH agent:

```
ssh-add ~/.ssh/id_ed25519
```

Or if you want to add all of the available keys under the default `.ssh` directory, simply run:

```
ssh-add
```

Notes for macOS User

If you're using macOS Sierra 10.12.2 or later, to load the keys automatically and store the passphrases in the Keychain, you need to configure your `~/.ssh/config` file:

```
Host *
  AddKeysToAgent yes
  UseKeychain yes
  IdentityFile ~/.ssh/id_ed25519
  IdentityFile ~/.ssh/id_rsa # Keep any old key
files if you want
```

Once the SSH config file is updated, add the private-key to the SSH agent:

```
ssh-add -K ~/.ssh/id_ed25519
```

Specifying Specific Key to SSH into a Remote Server

The SSH protocol already allows the client to offer multiple keys on which the server will pick the one it needs for authentication. However, we can also specify a specific private-key to use like so:

```
ssh -i ~/.ssh/id_ed25519 john@198.222.111.33
```

Or you can even add an entry to the `~/.ssh/config` file to configure these options:

```
Host awesome
  HostName 198.222.111.33
  User john
  IdentityFile ~/.ssh/id_ed25519
  IdentitiesOnly yes
```

Once it's saved, later you can SSH to your target host like this:

```
ssh awesome
```

Awesome right? 😎

Originally published at risanb.com on 29 November 2017.

Ssh Servers DevOps Security Ed25519